ARMY RESEARCH LABORATORY

Multiple Objective Evolution Strategies (MOES): A User's Guide to Running the Software

by James Lill and Anthony Yau

ARL-CR-0753 **November 2014**

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5069

ARL-CR-0753 **November 2014**

Multiple Objective Evolution Strategies (MOES): A User's Guide to Running the Software

James Lill
Engility Corporation, High Technology Services Group

Anthony Yau
Formerly with High Performance Technologies, Inc.

under contract

MIPR9BO47BW060

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2014	Final	October 2008–September 2009

4. TITLE AND SUBTITLE
Multiple Objective Evolution Strategies (MOES): A User's Guide to Running the Software

5a. CONTRACT NUMBER
MIPR9BO47BW060

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

2986. AUTHOR(S)
James Lill and Anthony Yau

5d. PROJECT NUMBER
GS04T01BFC006120

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Engility Corporation, High Technology Services Group, and
Formerly with High Performance Technologies, Inc.

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-WML-B
Aberdeen Proving Ground, MD 21005-5069

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ARL-CR-0753

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A user's guide for the parallel Multiple Objective Evolution Strategies (MOES) software package is presented. MOES employs a sophisticated self-adaptive evolutionary algorithm known as Evolution Strategies. The software can perform single objective optimization (with and without constraints) as well as multiple objective optimization using a fitness function based on Pareto dominance. The novel multiple-objective fitness function is computed using the concept of efficiency from Data Envelopment Analysis (DEA), a specialized application of linear programming. MOES is unique in combining a very flexible self-adaptive algorithm with a novel multiple-objective algorithm to compute Pareto fitness, all within a package that has been efficiently parallelized.

15. SUBJECT TERMS
Evolution strategies, optimizations, Pareto dominance, data envelopment analysis, fitness

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Betsy Rice
Unclassified	Unclassified	Unclassified	UU	32	19b. TELEPHONE NUMBER (*Include area code*) 410-306-1904

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures iv

List of Tables iv

1. Introduction 1

2. Example Analysis 1

3. *Input.in* Keywords 4

4. Output Files Generated by MOES 15

Appendix. Data Envelopment Analysis (DEA) Models 19

List of Symbols, Abbreviations, and Acronyms 25

Distribution List 26

List of Figures

Fig. 1 Plot demonstrating the convergence behavior for 9 independent evolutions of 1 MOES problem 2

Fig. 2 Plot showing the Pareto-efficient solutions in the elite population of a MOES calculation 3

Fig. 3 Plot showing the inefficient solutions in the elite population of a MOES calculation 4

List of Tables

Table Sample input file for a MOES calculation containing all input arguments 5

1. Introduction

The Multiple Objective Evolution Strategies (MOES) software package, developed in the Multiscale Reactive Modeling of Insensitive Munitions (MSRM-IM) Software Application Institute, implements algorithms for multiple objective parameter optimizations that join evolutionary strategies with Data Envelopment Analysis (DEA), a specialized application of linear programming. The MOES binary executable is compiled with parallelization by default and requires only one input file: **`Input.in`**. This report is a User Guide that discusses running MOES, although its principle use will be to fit Reax force field (ReaxFF) parameters. Because MOES drives the optimization but relies on the REAC program, the developer's original software, to run ReaxFF, all of the input files that are needed by REAC to fit a force field are also required. Descriptions of their functions and formats can be found in the REAC User Manual. Inside a directory with the MOES and REAC input files, the program is invoked with the standard Message Passing Interface (MPI) launch program. An example launch program is provided below.

```
mpirun –np 4 moes.x
```

MOES has no command-line arguments and can run with only 1 MPI task.

2. Example Analysis

This section of the User Guide emphasizes the most important data streams after a successful MOES calculation.

The file **`history.out`** contains a detailed chronology of what MOES is doing. If the calculation terminates unexpectedly, the first places to check are in standard error, in standard output, and in the end of the history file. If the calculation terminates normally (including stalled evolutions), then the file **`trajectory.out`** will probably provide the most useful information for the user. The columns in trajectory.out describe numerous data that are useful in determining whether the evolutions have converged. Columns labeled **`LPSOLVE-##`** list the number of times the lp_solve library generated a particular result. In most cases, result **`0`** (no error) is generated for each solution in each generation. A few instances under **`LPSOLVE- 2`** are also acceptable and indicate that an occasional infeasible linear program was generated.

The most sensitive measures of convergence are provided by the standard deviations used in Evolution Strategies to control mutations of the real variables. These are listed under the labels **`deviation-##`** for each real variable; an average of all standard deviations is given under the

label **STDave**. Typical converged results for 9 independent evolutions of one problem are shown in Fig. 1.

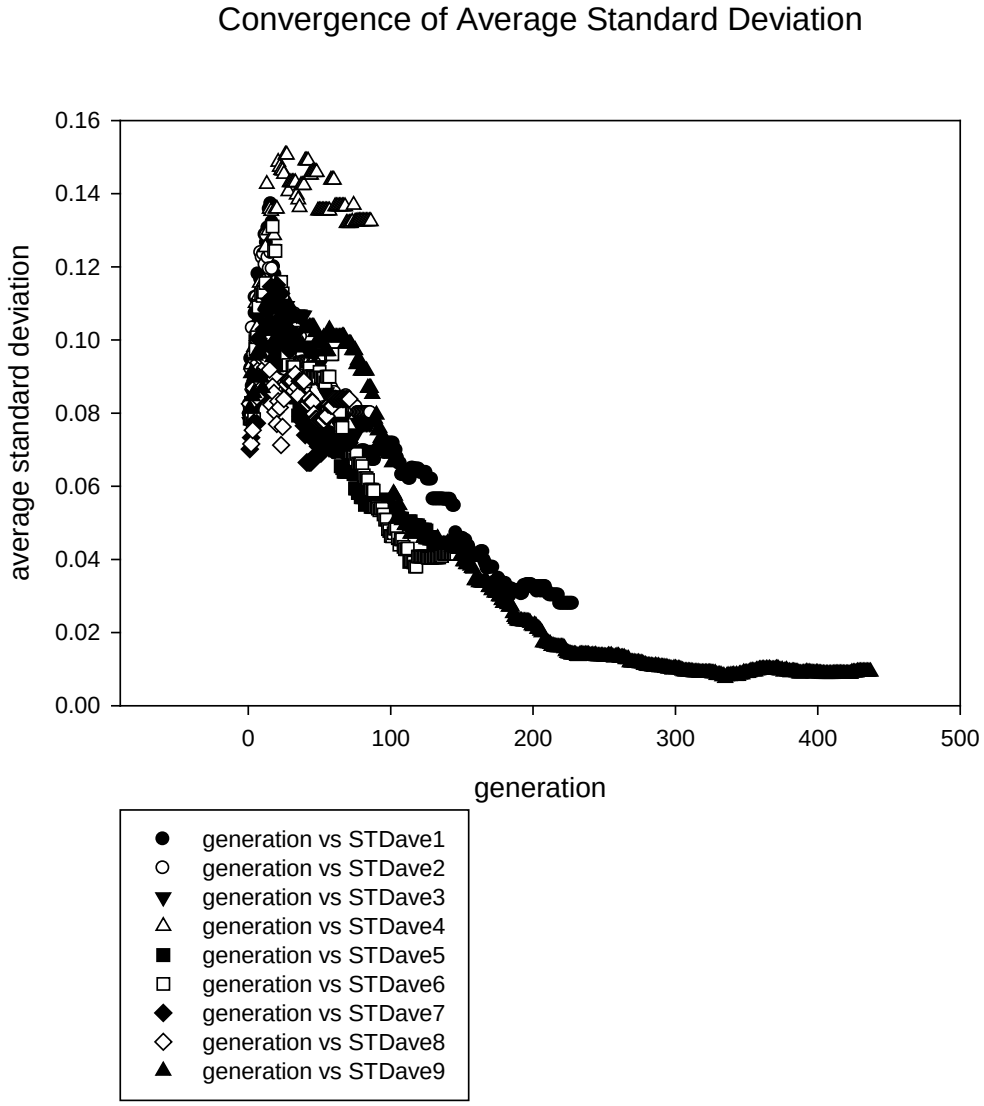

Fig. 1 Plot demonstrating the convergence behavior for 9 independent evolutions of 1 MOES problem

Several of the evolutions in the example do not converge. They are terminated early (e.g., Evolutions 4 and 8). However, common evolutionary behavior is to increase initial deviation as the algorithm examines phase space and then to decrease rapidly as the algorithm concentrates upon a local minimum. Graphs using data in the trajectory file should be constructed with either the generation or the number of objective function evaluations (**N_compute**) as the abscissa. The column labeled **N_Elite** lists the size of the elite population at each generation, and the column **infeasible** lists the number of solutions that violate constraints when performing constrained optimization with **problem_type = 1** (in the MOES input file). The entries under **Objective-##**, **variable-##**, and **deviation-##** refer to those values for the

best solution; in the case of multiple objective optimizations, these values are somewhat meaningless as they correspond to one of the solutions on the Pareto frontier. Assuming an evolution converges, the next file to examine is **`final.out`**. The Pareto-efficient solutions can be identified by those solutions whose efficiency is unity. (Using a scatter plot with the data organized in x-y pairs is most convenient.) Figure 2 shows a typical Pareto set.

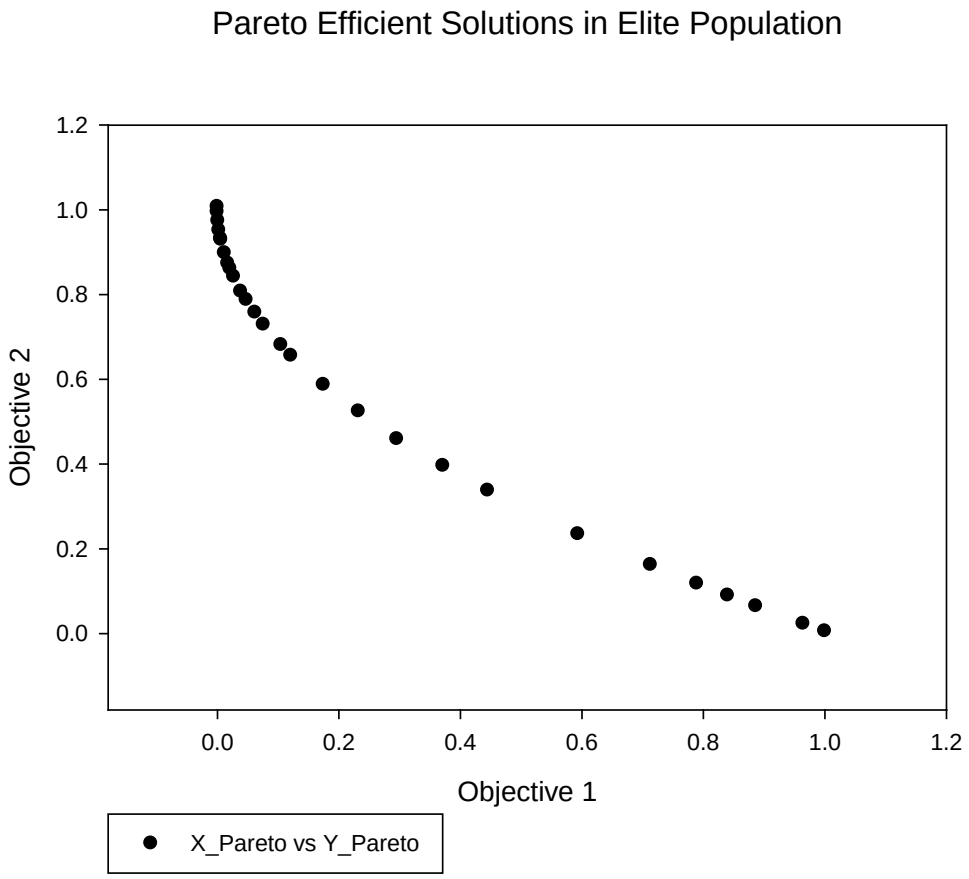

Fig. 2 Plot showing the Pareto-efficient solutions in the elite population of a MOES calculation

The remaining inefficient solutions in the elite population can be graphed in a similar manner for comparison. A typical plot for all elite-but-inefficient results is shown in Fig. 3.

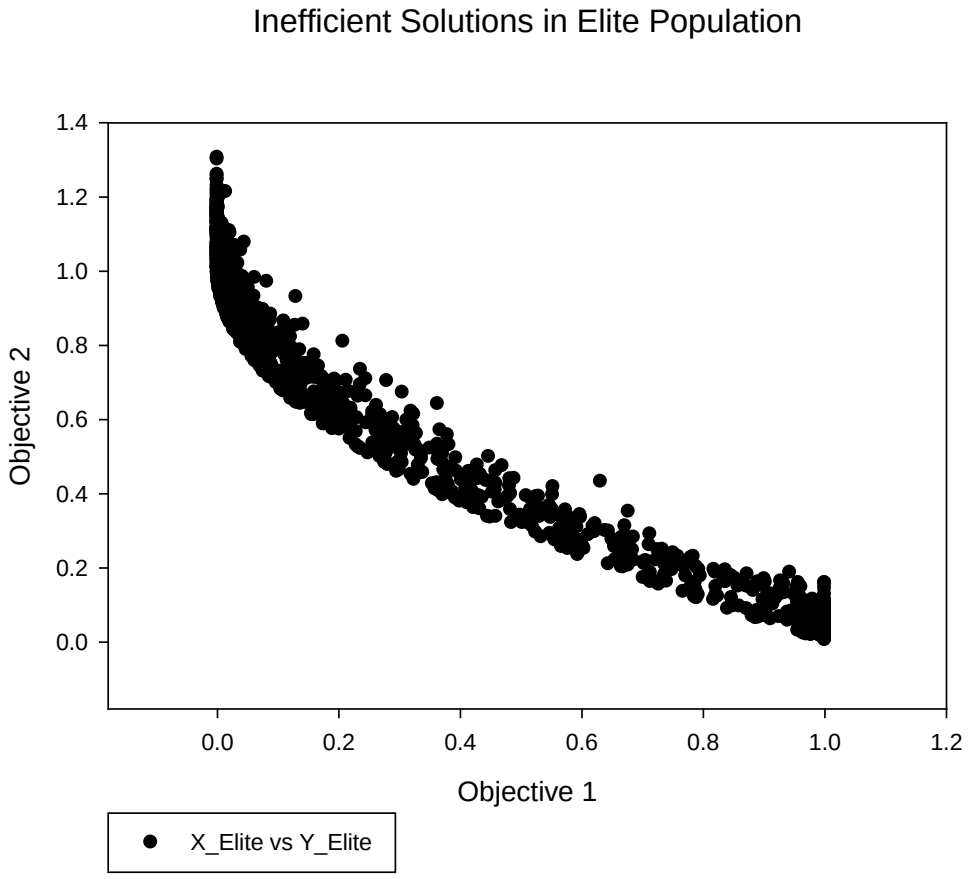

Fig. 3 Plot showing the inefficient solutions in the elite population of a MOES calculation

The solution variables for the efficient solutions—as well as those for the formally inefficient solutions whose efficiency is close to unity—are the candidate solutions that should be considered as possible optimal parameter sets for ReaxFF.

3. *Input.in* Keywords

A sample input file for multiple objective optimization of ReaxFF parameters is shown in the Table below. Values for all the input arguments must be specified in the order and format given below even if a certain input argument is irrelevant to the particular calculation being performed. The routine that reads the input file expects all the arguments given in the Table below.

Table Sample input file for a MOES calculation containing all input arguments

Line	Sample Input File
1	ReaxFF parameter optimization
2	problem_type = 0
3	I_Recombine = 0
4	I_Holdout = 0
5	I_Print = 0
6	print_Trajectory = 1
7	ITerminate = 1
8	N_Window = 50
9	Evolve_Stall_Tolerance = 0.001
10	Simplex_Stall_Tolerance = 0.001
11	h_width = 0.01
12	Objective = 1000
13	ranseed = 1
14	population_seed = 0
15	Iclone = 1
16	iRestart = 0
17	N_evolutions = 1
18	N_generations = 1000
19	I_Fit = 0
20	DEA_model = FDH-I
21	skip_LPSOLVE_errors = 1
22	DEA_PRINT = 0
23	N_agglomerate = 100
24	Limit_Elite = 10
25	strategy = mu,kappa,lambda
26	selection = truncation
27	mu = 50
28	kappa = 15
29	lambda = 450
30	rho = 2
31	N_tournament = 2
32	recombination = none
33	x_recombine_mode = sexual
34	x_recombine_operator = discrete
35	s_recombine_mode = panmictic
36	s_recombine_operator = intermediate
37	Use_Angles = 0
38	a_recombine_mode = panmictic
39	a_recombine_operator = intermediate
40	i_recombine_mode = sexual
41	i_recombine_operator = discrete

Table Sample input file for a MOES calculation containing all input arguments (continued)

Line	Sample Input File
42	p_recombine_mode = sexual
43	p_recombine_operator = intermediate
44	use_Amoeba = 0
45	use_ConGrad = 0
46	use_CnvxGlblUndr = 0
47	MAX_cluster = 5
48	Linear_Ranking = 1
49	Bias = 1.1
50	CGU_iterate = 0
51	nCGU = 244
52	CGU_distribution = Gaussian
53	CGU_anneal = 0
54	xCGU = 0.5

Line 1 is a character string that is printed at the top of the **history.out** file to describe the calculation. There is no particular format (other than having an 80-character limit). The remaining lines must all follow the format as shown above (e.g., in Line 12, **Objective =**) and must be entered in the same order. The input arguments are defined below.

problem_type = %d\n—This integer argument defines the optimization problem type. The possible values are 0 for multiple objective optimization, 1 for constrained single objective optimization, and 2 for unconstrained single objective optimization.

I_Recombine = %d\n—This integer argument sets the number of evolutions to skip before computing the DEA fitness. The possible values are 0 to perform a normal evolution, or >0 to skip the evolution proper and compute the DEA fitness of the combined solutions from I_Recombine independent evolutions.

I_Holdout = %d\n—This integer argument sets the number of evolutions to skip before computing the DEA fitness using a new training set containing holdout data, where holdout data are additional data that supplement the original training set data. The possible values are 0 to perform a normal evolution or to compute DEA efficiencies of **I_Recombine** independent evolutions, or >0 to skip the evolution and recompute the fitness of solutions from the **I_Recombine** independent evolutions using a new training set that contains holdout data.

I_Print = %d\n—This integer argument specifies whether the debug information is to be printed. The possibilities are 0 to suppress debug printing and 1 to print debug information.

print_Trajectory = %d\n—This integer argument specifies whether the convergence information for each evolution is to be printed. The possible choices are 0 to print data describing the convergence of each evolution and 1 to suppress this printing.

ITerminate = %d\n—This integer argument defines the criteria that are to be used to terminate the calculation. The possible choices are 0, 1, and 2. Setting 0 results in each separate evolution terminating only after all the specified generations have been evolved. Setting 1 results in each separate evolution terminating after it has been determined that the evolution has stalled. In unconstrained single objective optimizations, an evolution is considered to be stalled when the current best objective differs from the averaged best objective over the past **N_Window** generations by an amount less than **Evolve_Stall_Tolerance**.

In constrained single objective optimizations, an evolution is considered to be stalled when the current best fitness differs from the averaged best fitness over the past **N_Window** generations by an amount less than **Evolve_Stall_Tolerance**. In multiple objective optimizations, an evolution is considered to be stalled when each of the current best objectives averaged over the solutions on the Pareto frontier differs from the corresponding averaged best objectives over the past **N_Window** generations (again averaged over the solutions on the Pareto frontier) by an amount less than **Evolve_Stall_Tolerance**. Thus each evolution can evolve for a different number of generations. Setting 2 results in each separate evolution terminating after it has been determined that the evolution has stalled; however, in this case, different criteria are used to determine when the evolution has stalled. In both unconstrained and constrained single objective optimizations, as well as in multiple objective optimizations, when the average number of generations the solutions in the elite population have spent in the elite population exceeds **kappa** (see kappa discussion below), the evolution is considered to be stalled.

N_Window = %d\n—This integer argument sets the number of most recent generations to use for moving averages in order to determine when a particular evolution has stalled (see the discussion of **ITerminate** above). The typical choices range from 10 to 100.

Evolve_Stall_Tolerance = %lf\n—This real-valued argument specifies the stall tolerance. When **ITerminate = 1**, this determines how long each evolution is allowed to proceed.

Simplex_Stall_Tolerance = %lf\n—This real-valued argument specifies the stall criterion when using Downhill Simplex optimization, which is currently disabled.

h_width = %lf\n—This real-valued argument specifies the amount by which the equality constraint can be violated. When performing constrained single objective optimizations, each equality constraint is transformed into a pair of greater-than **h_width** and less-than **h_width** constraints.

Objective = %d\n—This integer argument, in combination with the **problem_type** argument, defines the particular objective function to be used. MOES contains a number of hard-

coded objective functions in the routine **`void F_objective`** in the source file **`MOES.cpp`**. For ReaxFF objectives, the integer value is 1000.

`ranseed = %d\n`—This integer argument defines how the random number seed is selected. Setting 0 results in using the same random number seed each time the program is run, which is useful for debugging. Setting 1 results in different random number seeds being used each time the program is run, which is required for production runs.

`population_seed = %d\n`—This integer argument defines how the initial population is generated. The possible values are 0, 1, and 2. The default value 0 indicates that the initial population is determined by generating uniform variates within the finite bounds that must be set for each variable (i.e., the initial population is randomly generated). Setting 1 fills the initial population with a single solution defined by the **`variable_seed`** array specified in the objective function. The value 2 is reserved for an option in which variables for seed solutions must be read off disk. (Option 2 has not yet been implemented.)

`Iclone = %d\n`— This integer argument specifies whether duplicate solutions are to be eliminated from the elite population. The possible choices are 0 and 1. Setting 1 eliminates nearly duplicate solutions (clones) from the elite population. An L2-norm distance is computed in parameter space between each pair of solutions in the elite population and tested against the tolerance **`Dclone`**, which is hardcoded in each objective function.

`iRestart = %d\n`—This integer argument sets the number of completed evolutions. When **`iRestart`** is >0, the file **`esdata.out`** is used to initialize the elite population and the loop over evolutions is started at evolution **`N+1`**. This feature can be used to gather together evolutions from multiple machines by collating and renumbering the Evolution and Solution indices in **`esdata.out`**. Currently, calculations cannot be restarted in the middle of an evolution.

`N_evolutions = %d\n` —This integer argument sets the total number of evolutions to be performed. If **`iRestart`** is >0, then only the remaining evolutions are performed. The final analysis of best solutions, written to file **`final.out`**, is performed only after all of the evolutions have been completed.

`N_generations = %d\n`—This integer argument sets the maximum number of generations in each evolution. If **`ITerminate = 0`**, then each evolution continues for this maximum number of generations. See the discussion of the input argument **`ITerminate`**.

`I_Fit = %d\n`—This integer argument defines how fitness is computed for the various DEA models. The possible choices are 0, 1, and 2. This controls how fitness is computed for the various DEA models when multiple objective optimization is performed. Setting 0 specifies the fitness of a multiple objective solution using so-called weak DEA efficiency for the radial DEA models or 1 plus the sum of the slacks for the additive DEA models. MOES adds 1 to the sum of the slacks for the additive models so that unit fitness implies that the solution lies on DEA

frontier for both the radial and the additive models. Setting 1 defines the fitness of a solution using a nonlinear combination of weak efficiency and slacks; this setting can only be employed with the radial DEA models. Setting 2 defines the fitness of a solution using one plus the sum slacks; this setting can only be employed with the additive DEA models. The developers recommend using one of the radial DEA models (see the discussion for the input argument **`DEA_model`**) and setting **`I_Fit = 1`**.

`DEA_model = %s\n`—This string argument defines the type of DEA model that is to be used. MOES uses DEA to compute fitness for solutions to multiple objective optimizations. In DEA, a unique linear program must be solved for each solution in the evolving population. The number of rows in the linear program is comparable to the number of solutions in the evolving population; the number of rows is comparable to the number of objectives to be optimized. The possible choices are **`CCR-I`**, **`CCR-O`**, **`BCC-I`**, **`BCC-O`**, **`FDH-I`**, **`FDH-O`**, **`ADD-CRS`**, **`ADD-VRS`**, and **`ADD-FDH`**. These abbreviations stand for the input- (**`-I`**) and output-oriented (**`-O`**) Charnes-Cooper-Rhodes (**`CCR-`**), Banker-Charnes-Cooper (**`BCC-`**), and Free Disposal Hull (**`FDH-`**) radial models and the constant returns to scale (**`ADD-CRS`**), variable returns to scale (**`ADD-VRS`**), and free disposal hull (**`ADD-FDH`**) additive models. All of the radial models (CCR, BCC, and FDH) are invariant to different scale factors applied to the different objectives; that is, despite any differences in the scales of the objectives, these models should give the same ordering when comparing the fitness of the solutions. By contrast, the additive models are *not* invariant to changes in scale.

Furthermore, the ADD-FDH model is experimental and has not been verified. The developers recommend using the radial models. The CCR and BCC models assume that the Pareto frontier is convex, thus CCR-efficient and BCC-efficient solutions may not find all the Pareto-efficient solutions for a particular problem. The FDH model makes no such assumption. The FDH-efficient solutions will always be Pareto efficient; however, the FDH model takes longer to compute because it involves solving an integer linear program. The choice of whether to use input- or output-oriented models depends on the type of objectives to be optimized. In the language of DEA, objectives that are to be maximized are called *outputs* and those that are to be minimized are called *inputs*. For example, minimizing the error between classical ReaxFF energies and ab initio energies in the training set uses input-oriented models with a phony output objective that is created within the program.

`skip_LPSOLVE_errors = %d\n`—This integer argument specifies whether MOES will terminate when encountering an infeasible linear program. The possible choices are 0 and 1. If the argument is set to 0, then the multiple objective optimization will terminate upon encountering an infeasible linear program when computing Pareto fitness and print diagnostic information. If the argument is set to unity, then a bad fitness value will be assigned to the solution that generated an infeasible linear program and the evolution will continue. The developers recommend ignoring LP_SOLVE errors when optimizing ReaxFF parameters.

DEA_PRINT = %d\n—This integer argument specifies whether debug information generated from the linear program is to be printed. The possible choices are 1 to print the debug information and 0 to suppress this printing. Debug printing generates considerable output and should only be used for debugging.

N_agglomerate = %d\n—This integer argument sets the maximum number of solutions on the Pareto frontier as a result of agglomerative clustering. In some problems, it is possible to generate so many Pareto-efficient solutions that the Pareto frontier becomes crowded. It might become necessary to perform agglomerative clustering of the Pareto-efficient solutions into **N_agglomerate** clusters that will still cover the Pareto frontier as well as the original solutions.

Limit_Elite = %d\n—This integer argument sets the maximum number of solutions in the external (non-evolving) elite population during single objective optimizations. The elite population is the collection of the best solutions obtained during the course of an evolution.

strategy = %s\n—This string argument defines the type of Evolution Strategy to employ. The possible choices are **mu,kappa,lambda,rho**, **mu,kappa,lambda**, **mu+lambda**, and **mu,lambda** using the standard notation of Bäck for Evolution Strategies. **mu,lambda** is perhaps the most commonly employed Evolution Strategy. In this case, the **mu** parents die off at the end of each generation, the **lambda >= mu** offspring are then ordered by fitness, and the best **mu** solutions retained as parents for the next generation. This strategy is called non-elite because, even if one of the parents happens to be a particularly good solution, it is killed off immediately after it reproduces and the evolving population loses its genome. The opposite case, in which a parent can in principle live forever, is provided by the **mu+lambda** strategy. Here, in each generation, the entire evolving population of **mu** and **lambda** solutions is ordered by fitness, and the best **mu** solutions are retained as parents for the next generation.

The consensus opinion is that **mu+lambda** strategies are considerably more prone to premature convergence than are **mu,lambda** strategies. Intermediate between these 2 extremes is the **mu,kappa,lambda** strategy, in which each solution is allowed to live for no more than **kappa** generations. The flexibility of this strategy allowed for some of the most efficient solutions to the test problems in the **Input_Files** directory. The developers recommend using **mu,kappa,lambda**. The **mu,kappa,lambda,rho** strategy is only used in conjunction with **recombination = diagonal_crossover**. In this case, **rho** gives the number of parents that are recombined using diagonal crossover. See the discussion below for **recombination**.

selection = %s\n—This string argument defines the type of selection to employ. The possible choices are **truncation** and **tournament**. Truncation is the standard method of selection in Evolution Strategies. The solutions in the evolving population are ordered according to fitness, and the best **mu** solutions are retained as parents for the next generation. If users suspect that evolutions are ending prematurely, then tournament selection might be beneficial

because it has a much lower selective pressure. In this case **N_tournament** solutions are chosen at random from the evolving population; the solution with the highest fitness is removed from the evolving population and is chosen as a parent for the next generation. Another tournament is then held.

mu = %d\n—This integer argument sets the number of parents in the evolving population using the standard notation of Bäck for Evolution Strategies. MOES initializes each evolution with **mu** parents according to the **population_seed** argument. A general rule of thumb is that the number of parents should be roughly 2 times the number of evolving parameters.

kappa = %d\n—This integer argument sets the maximum lifetime (in generations) of solutions in the evolving population.

lambda = %d\n—This integer argument sets the number of offspring in the evolving population. The time to compute each generation depends entirely on this input argument since it sets the number of times the objective function must be called in the loop over generations. A general rule of thumb is that the number of offspring should be roughly 8 times the number of evolving parameters.

rho = %d\n—This integer argument sets the number of parents used with diagonal crossover. **rho - 1** crossover points are chosen at random in the genome (parameter space) and the parameters of the offspring are chosen from those of **rho** parents chosen at random. In some cases, this crossover is thought to preserve more good genes than the more commonly used uniform crossover described below.

N_tournament = %d\n—This integer argument sets the number of solutions used in each tournament selection. The selected solutions are not returned to the evolving population.

recombination = %s\n—This string argument defines the type of recombination method to employ, if any. The possible choices are **uniform_crossover**, the standard recombination method when performing Evolution Strategies; **diagonal_crossover**, a less-commonly used recombination method in Evolution Strategies that is described above; and **none**, i.e., perform mutation-only Evolution Strategies without a recombination method. In the standard uniform crossover, each parameter of each offspring is chosen from one or more parents according to the various modes and operators given below. There are currently 2 schools of thought in the evolutionary algorithm community regarding recombination: one school says recombination only screws things up and should not be used in general; the other school says recombination is generally effective if its application is sufficiently flexible. The developers remain agnostic in this debate. MOES does allow users to experiment with multiple recombination options, in which case the developers recommend using **uniform_crossover**.

x_recombine_mode = %s\n—This string argument defines the mode of uniform crossover on the floating-point parameters being varied. The choices are **sexual** and **panmictic**. In the

sexual mode, only 2 parents chosen at random from the **mu** parents can contribute genetic material to the offspring; many parents can contribute in the panmictic mode. One parent is chosen at random from the **mu** parents and then a different partner is chosen at random from the **mu - 1** remaining parents for each parameter to be recombined. Thus, genetic material from 2 parents is transferred to each parameter, but one of the parents is different each time.

x_recombine_operator = %s\n—This string argument defines the uniform crossover operator on the floating-point parameters being varied. The possible choices are **discrete**, **intermediate**, and **general**. If the operator is **discrete**, then the parameter of the offspring is chosen to be one of the parameters of the 2 parents involved in recombination for that particular real parameter. If the operator is **intermediate,** then the parameter of the offspring is chosen to be the average of the parameters of the 2 parents involved in recombination for that particular real parameter. If the operator is **general,** then the parameter of the offspring is chosen to be a stochastic linear combination of the parameters of the 2 parents involved in recombination for that particular real parameter.

The following recombination modes and operators follow a similar pattern.

s_recombine_mode = %s\n—This string argument defines the uniform crossover mode that is to be performed on the standard deviations, i.e., the strategy parameters that control the mutation of the real parameters. The possible choices are **sexual** and **panmictic**.

s_recombine_operator = %s\n—This string argument defines the uniform crossover operator for the standard deviations. The possible choices are **discrete**, **intermediate**, and **general**.

Use_Angles = %d\n—This integer argument specifies whether correlated mutations are to be performed. The possible choices are 0 and 1. Setting 1 enables covariant angles when performing correlated mutations, which require copious amounts of memory but can potentially converge evolutions much more rapidly.

a_recombine_mode = %s\n—This string argument defines the uniform crossover mode for the covariant angles that control the correlated mutation of the parameters to be varied. The possible choices are **sexual** and **panmictic**.

a_recombine_operator = %s\n—This string argument defines the uniform crossover operator for the covariant angles. The possible choices are **discrete**, **intermediate**, and **general**.

i_recombine_mode = %s\n—This string argument defines the uniform crossover mode for the varying integer parameters. The possible choices are **sexual** and **panmictic**.

i_recombine_operator = %s\n—This string argument defines the uniform crossover operator for the varying integer parameters. The possible choices are **discrete**,

intermediate, and **general**. A nearest-integer function is employed to ensure that the result of intermediate or general recombination is an integer.

p_recombine_mode = %s\n—This string argument defines the uniform crossover mode for the mutation probabilities, which are the strategy parameters that control the mutation of the varying integer parameters. The possible choices are **sexual** and **panmictic**.

p_recombine_operator = %s\n—This string argument defines the uniform crossover operator for the mutation probabilities. The possible choices are **discrete**, **intermediate**, and **general**.

use_Amoeba = %d\n—This integer argument defines how the downhill simplex (DHS) algorithm is to be employed and how it is only valid for single objective optimization. (Amoeba is the name of the DHS implementation in Numerical Recipes. Because Numerical Recipes software cannot be distributed, this option has been **disabled**, but the keyword must still be present.) The possible choices are 0, 1, 2, and 3. If 0, DHS is not used. If 1, DHS is applied to the elite population at the end of each evolution. The effect is that the elite population will be clustered around a dense set of local minima; using them as an initial simplex may find a better solution. If 2, DHS is applied to the elite population at the end of each evolution as well as to the clustered results of all of the quenched elite populations from all the evolutions. In other words, the best results from applying DHS to the elite population after each evolution are clustered and the DHS is applied a final time to try to find a better solution. The effect is that each evolution has found a local optimum among some presumed dense set of local minima and that a final application of the DHS might find one better.

Options 1 and 2 represent hybrid algorithms in which the DHS refines the optima obtained by Evolution Strategies. This hybrid algorithm may in fact reduce the number of function evaluations if the switch is made at the correct time. If **use_Amoeba = 3**, then evolutionary optimization is skipped entirely and the DHS is applied directly to the best initialized population of solutions. This option can be used as a sanity test for complicated multimodal problems. Whenever the optimum cannot be reached by a local search, the evolutionary results should be far superior. The size of the simplex used in the algorithm is dictated by the dimensionality of the parameter space.

use_ConGrad = %d\n—This integer argument defines how the conjugate gradient algorithm is to be employed; it is only valid for single objective optimization. (Conjugate gradients have also been **disabled** due to Numerical Recipes' licensing restrictions.) The possible choices are 0, 1, and 2. If 0, the conjugate gradient algorithm is not employed. If 1, then the conjugate gradient algorithm is used to quench the final parent population at the end of each evolution. The elite population then consists of the best quenched solutions obtained from all the evolutions performed to date. This is a hybrid algorithm that can save function evaluations if the switch between Evolution Strategies and the conjugate gradient algorithm is made at the right time. If 2, then Evolution Strategies is skipped entirely and the conjugate gradient algorithm is

applied directly to the initialized population of solutions. Again, this option can be used as a sanity test for complicated multimodal problems. Whenever the optimum cannot be reached by a local search, the evolutionary results should be far superior.

`use_CnvxGlblUndr = %d\n`—This integer argument defines how the Convex Global Underestimator algorithm (CGU) is to be employed. The possible choices are 0, 1, and 2. If 0, CGU will not be used. If 1, CGU is used on the final parent population. If 2, Evolution Strategies are skipped entirely and CGU is applied to the initialized population. CGU is a single objective algorithm originally developed for studying protein folding. It assumes that the energy surface looks like the proverbial bumpy funnel. Given a set of points in parameter space, the algorithm quenches the objective at each point and then constructs a quadratic function that underestimates the energy at each point. The coefficients of the quadratic polynomial are obtained using linear programming. The CGU was implemented with the LP_SOLVE library and verified against an older version of the program. The code has been upgraded with calls to the latest LP_SOLVE library, but it has not yet been incorporated into the latest version of MOES. This option has not yet been implemented. Several of the parameters relating to CGU at the end of the current input file may be deleted after CGU has been implemented.

`MAX_cluster = %d\n`—This integer argument sets the maximum number of clusters that the code attempts to form from the best quenched solutions from each evolution before final application of the DHS algorithm. It is used in conjunction with **`use_Amoeba = 2`** and must be less than **`N_evolutions`**.

`Linear_Ranking = %d\n`—This integer argument specifies whether linear ranking is to be used in choosing parents for recombination. Normally, parents are chosen at random using uniform variants, i.e., parents are chosen in a non-biased fashion and parents with high fitness are just as likely to reproduce as parents with low fitness. In linear ranking, the parents are ordered by fitness; a linear bias is applied when choosing parents to recombine. The possible choices are 0, not to use linear ranking; and 1, to use linear ranking.

`Bias = %lf\n`—This real-valued argument specifies the value of the bias used in linear ranking; it must be > 0.0 and < 2.0.

`CGU_iterate = %d\n`—This integer argument has not been implemented.

`nCGU = %d\n`—This integer argument has not been implemented.

`CGU_distribution = %s\n`—This string argument has not been implemented.

`CGU_anneal = %d\n`—This integer argument has not been implemented.

`xCGU = %lf\n`—This real-valued argument has not been implemented.

4. Output Files Generated by MOES

history.out

The **history.out** file is the principal output file that contains a history of the calculation. In parallel calculations, there are similar files, **history_1.out**, **history_2.out**, etc., created by each processing element. Much of the history of a calculation is documented with output such as the following:

```
begin generation 97, evolution 66, N_compute = 8175
          Population[100].efficiency = +1.000000e+00,
Population[100].max_slack = +0.000000e+00
          Population[100].fitness = +1.000000e+00, Population[100].gvalue =
+1.094935e+00
          Population[100].objective[1] = +6.117722e-01,
Population[100].slack[1] = +0.000000e+00
          Population[100].objective[2] = +2.764909e-01,
Population[100].slack[2] = +0.000000e+00
          Population[100].objective[3] = +1.000000e+00,
Population[100].slack[3] = +0.000000e+00


 ... computing DEA efficiency of parents and children in evolving population,
case 2, Ndmu = 100
 ... using CCR-I model, ipath = 1

 arrays dimensioned for DEAsolve

 input & output variables assigned for DEAsolve

 ... call DEAsolve: ipath = 1, Ndmu  = 100, Nx  = 2, Ny  = 1, dea_print  = 0

 DEAsolve returned Ierror = 0, Kerror = 0, Perror = 0

 analyzing best current solutions using DEA, N_Elite = 12


 ... computing DEA efficiency of elite population, case 3, Ndmu = 12
 ... using CCR-I model, ipath = 1

 arrays dimensioned for DEAsolve

 input & output variables assigned for DEAsolve

 ... call DEAsolve: ipath = 1, Ndmu  = 12, Nx  = 2, Ny  = 1, dea_print  = 0

 DEAsolve returned Ierror = 0, Kerror = 0, Perror = 0
 new total Pareto efficient solutions, N_Elite = 9
```

Some statistics are printed after each evolution:

```
mean and standard deviation of 1020 elite solutions:
 mean = +9.136540e-01, standard deviation = +5.695545e-02, median =
+9.105912e-01
 best elite fitness = +1.000000e+00, worst elite fitness = +7.435670e-01


 average number of generations evolved = 149


 average simplex iterations per generation = 1415

 Exiting Server
```

If the calculation ends unexpectedly, check **history.out** for error messages and diagnostic information. Each MPI task records the solutions with the lowest L1-norm (Manhattan distance) and L2-norm (Euclidean distance) in objective space. As the evolution proceeds, the moment a solution with a new smallest error is discovered, it prints the parameter values and the norm to its **history_###.out** file. This feature is only meant for walking away from a crashed evolution with at least one force field; it should not be used to attempt to initialize an elite population because the solutions have not been filtered with an envelopment analysis or a clone detection algorithm.

trajectory.out

The **trajectory.out** file is used to study the convergence of the evolutionary optimization. The results of the best solution obtained in each generation are printed, including fitness, objective(s), variables, and standard deviations. Convergence can be ascertained by graphing various results versus graphing either the generation or the total number of objective function evaluations. Some of the diagnostics available include the number of elite solutions in each generation (**N_Elite**), the average age of the parents (**mu_Age**), and the average standard deviation of the parents (**STDave**). The latter quantity should approach zero as the algorithm converges. Typically, the quantity will increase slightly as the algorithm adjusts its strategy parameters to explore more parameter space; then it should drop rapidly as the algorithm concentrates on a local optimum.

pareto.out

The **pareto.out** file contains the DEA-efficient solutions in each generation as well as diagnostics for terminating the evolution. See the discussion of the input variables **ITerminate** and **N_Window** above. If an analytic Pareto frontier is known, it will be printed at the top of the file. In multiple objective optimizations, any one of the DEA-efficient solutions that appear in **pareto.out** can be printed in **trajectory.out**.

final.out

The **final.out** file shows all of the solutions obtained by combining the results of the final elite populations from each independent evolution. The solutions are ordered by their DEA-

efficiencies, which are obtained by performing the appropriate envelopment analysis of the combined results. In addition to efficiencies and objectives, all of the variables and standard deviations are also printed as well as is additional information, such as the projection of the solutions onto the Pareto frontier. This file will be empty unless the calculation proceeds to the final analysis of all evolutions.

amoeba.out

The **amoeba.out** file contains information about the calculations performed by the DHS algorithm whenever **amoeba** is invoked. See the discussion of input variables **use_Amoeba** and **Simplex_Stall_Tolerance** above.

lpsolve.out

The **lpsolve.out** file contains detailed information of the linear programs solved by the **LPSOLVE** package. It is controlled by the input variable **DEA_PRINT** and should only be invoked when debugging. Much of the output is generated by prints within **LPSOLVE** itself. Note that MOES solves many linear programs in each generation so a great deal of output can be generated with this option. The files **lpsolve_1.out**, **lpsolve_2.out**, etc., are created by each process if there are enough LP files to be distributed across all MPI tasks.

esdata.out

The **esdata.out** file is used to restart calculations (see the discussion of the input variable **iRestart**). The best solutions obtained after each evolution are combined into an array of structures called Combine[]. These structures are appended onto and read from **esdata.out**. Thus, when a calculation is restarted, it reads in Combine[] and begins at the next evolution.

best.out, parents.out, offspring.out

The **best.out**, **parents.out**, and **offspring.out** files contain detailed information of the best, parent, and offspring solutions obtained after each generation. They are controlled by the input variable **I_Print** and should only be used when debugging.

test.out

The **test.out** file is a catch-all output stream for debugging. It is written to when **I_Print = 1**. Currently, it records when solutions are written to or read from esdata.out.

RNG_test.out

The **RNG_test.out** file is a catch-all output stream for initializing the pseudo-random number generator.

INTENTIONALLY LEFT BLANK.

Appendix. Data Envelopment Analysis (DEA) Models

We summarize a series of DEA models for the p^{th} solution, i.e., the linear programs necessary to solve for the DEA efficiency of the p^{th} parameter set, θ_p, where $1 \le p \le N$ and where N is the number of solutions. For simplicity, we only summarize the input-oriented problems because these are set up to minimize objectives and these are what we have used while minimizing errors in the training set for Reax force field **(**ReaxFF) parameters. In our problems, the various errors we wish to minimize are treated as input variables in the context of DEA.

Input-Oriented Objective (DEA ratio): $\theta_p = \dfrac{\sum_{i=1}^{I} u_{pi} X_{pi} - u_0}{\sum_{j=1}^{J} v_{pj} Y_{pj}}$

minimize inputs, maximize outputs $\Rightarrow \min\left[\theta_p\right]$

I Input Variables: X_{pi}

J Output Variables: Y_{pj}

I Dual (multiplier) Solution Input Weights: u_{pi}

J Dual (multiplier) Solution Output Weights: v_{pj}

Because DEA demands that there be both input and output variables, we define a single constant output variable $Y_{p1} = 1$ for each solution (parameter set) p. The inputs X_{pi} then contain the various errors from the ReaxFF training set that Multiple Objective Evolution Strategies (MOES) tries to minimize. However, we have written out the DEA problems in the general case where there are multiple inputs and outputs. Formally, the DEA efficiency is defined in terms of the solutions to the dual linear program in multiplier form but, for numerical reasons, we solve the primal linear programs in envelopment form as indicated below. Pivoting in the tableaux is much more stable when there are more columns than rows, as is the case with the envelopment models. We can always recover the dual multipliers (weights) after obtaining the envelopment solution.

CCR-I

The input-oriented Charnes-Cooper-Rhodes (CCR) Model (Phase 1, weak efficiency) assumes the Pareto frontier to be convex.

Input-Oriented CCR Phase 1

objective: $\min\left[\theta_p\right]$

constraints: $0 \le \theta_p X_{pi} - \sum_{n=1}^{N} \lambda_{pn} X_{ni}$

constraints: $0 \ge Y_{pj} - \sum_{n=1}^{N} \lambda_{pn} Y_{nj}$

variable: $0 \le \theta_p$

variables: $0 \le \lambda_{pn}$

solution: $\hat{\theta}_p = \min\left[\theta_p\right]$

$1 \le i \le I =$ # inputs…$1 \le j \le J =$ # outputs…$1 \le n, p \le N =$ # solutions

tableau dimension: $N+1$ columns, $I+J$ rows

Phase 2 (max slack) then uses the solution of Phase 1 as a parameter:

Input-Oriented CCR Phase 2

objective: $\max\left[\sum_{i=1}^{I} s_{pi}^{-} + \sum_{j=1}^{J} s_{pj}^{+}\right]$

constraints: $\hat{\theta}_p X_{pi} - \sum_{n=1}^{N} \lambda_{pn} X_{ni} = s_{pi}^{-}$

constraints: $Y_{pj} - \sum_{n=1}^{N} \lambda_{pn} Y_{nj} = -s_{pj}^{+}$

parameter: $\hat{\theta}_p = \min\left[\theta_p\right]$

variables: $0 \le \lambda_{pn}$

slacks: $s_{pi}^{-} \ge 0, s_{pj}^{+} \ge 0$

solution: $\hat{s}_p = \sum_{i=1}^{I} s_{pi}^{-} + \sum_{j=1}^{J} s_{pj}^{+}$

$1 \le i \le I =$ # inputs…$1 \le j \le J =$ # outputs…$1 \le n, p \le N =$ # solutions

tableau dimension: N columns, $I+J$ rows

BCC-I

The input-oriented Banker-Charnes-Cooper (BCC) Model, (Phase 1, weak efficiency) adds a single convexity constraint to CCR. The Pareto frontier is assumed to be convex.

Input-Oriented BCC Phase 1

objective: $\min\left[\theta_p\right]$

constraints: $0 \le \theta_p X_{pi} - \sum_{n=1}^{N} \lambda_{pn} X_{ni}$

constraints: $0 \ge Y_{pj} - \sum_{n=1}^{N} \lambda_{pn} Y_{nj}$

constraint: $1 = \sum_{n=1}^{N} \lambda_{pn}$

variable: $0 \le \theta_p$

variables: $0 \le \lambda_{pn}$

solution: $\hat{\theta}_p = \min\left[\theta_p\right]$

$1 \le i \le I =$ # inputs…$1 \le j \le J =$ # outputs…$1 \le n, p \le N =$ # solutions

tableau dimension: $N+1$ columns, $I+J+1$ rows

Phase 2 (max slack) uses the solution to Phase 1 as a parameter:

Input-Oriented BCC Phase 2

objective: $\max\left[\sum_{i=1}^{I} s_{pi}^{-} + \sum_{j=1}^{J} s_{pj}^{+}\right]$

constraints: $\hat{\theta}_p X_{pi} - \sum_{n=1}^{N} \lambda_{pn} X_{ni} = s_{pi}^{-}$

constraints: $Y_{pj} - \sum_{n=1}^{N} \lambda_{pn} Y_{nj} = -s_{pj}^{+}$

constraint: $1 = \sum_{n=1}^{N} \lambda_{pn}$

parameter: $\hat{\theta}_p = \min\left[\theta_p\right]$

variables: $0 \le \lambda_{pn}$

slacks: $s_{pi}^{-} \ge 0, s_{pj}^{+} \ge 0$

solution: $\hat{s}_p = \sum_{i=1}^{I} s_{pi}^{-} + \sum_{j=1}^{J} s_{pj}^{+}$

$1 \le i \le I =$ # inputs…$1 \le j \le J =$ # outputs…$1 \le n, p \le N =$ # solutions

tableau dimension: N columns, $I+J+1$ rows

FDH-I

The input-oriented Free Disposal Hull (FDH) Model can be obtained from the input-oriented BCC phase 1 model by demanding that the variables be integers. In effect $\left(0 \le \lambda_{pn}\right)$ is replaced by $\left(\lambda_{pn} = \text{integer}\right)$. No assumption of convexity of the Pareto frontier is made.

Input-Oriented FDH

objective: $\min\left[\theta_p\right]$

constraints: $0 \le \theta_p X_{pi} - \sum_{n=1}^{N} \lambda_{pn} X_{ni}$

constraints: $0 \ge Y_{pj} - \sum_{n=1}^{N} \lambda_{pn} Y_{nj}$

constraint: $1 = \sum_{n=1}^{N} \lambda_{pn}$

variable: $0 \le \theta_p$

variables: $\lambda_{pn} = \text{integer}$

solution: $\hat{\theta}_p = \min\left[\theta_p\right]$

$1 \le i \le I =$ # inputs…$1 \le j \le J =$ # outputs…$1 \le n, p \le N =$ # solutions

tableau dimension: $N+1$ columns, $I + J + 1$ rows

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

BCC	Banker-Charnes-Cooper
CCR	Charnes-Cooper-Rhodes
CGU	Convex Global Underestimator
DEA	Data Envelopment Analysis
DHS	downhill simplex
FDH	Free Disposal Hull
MOES	Multiple Objective Evolution Strategies
MPI	Message Passing Interface
MSRM-IM	Multiscale Reactive Modeling of Insensitive Munitions
ReaxFF	Reax force field

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

1 (PDF) DIR USARL RDRL WML B J LILL A YAU